Using Rationale to Document Designs

Good Design Documentation
- Documentation has an important role in SW Eng
- There are many standards and methods for writing documents which typically specify their structure
- However, within a given document structure, it is still easy to write bad to useless documentation

Is it good documentation?
- A critical issue is the **quality** of the writing
  - have you identified the issues which need explanation?
  - are the given explanations clear and succinct?

Good for Whom?
- As with SW quality, when considering the **quality of documentation** we must take into account the audience who will be reading the documents.
  - Who is the document for?
  - What information will they be looking for?
  - How will they be using the document?

Rationale Overview
- Rationale is the justification of decisions
- Rationale is critical in two areas: it supports improved decision making and knowledge capture
- Rationale is important when designing or updating (eg. maintaining) the system and when introducing new staff

Rationale helps deal with change
- Improve maintenance support
  - Provide maintainers with design context
- Improve learning
  - New staff can learn the design by replaying the decisions that produced it
- Improve analysis and design
  - Avoid duplicate evaluation of poor alternatives
  - Make consistent and explicit trade-offs
Rationale Activities

- Rationale includes the **issues** that were addressed, the alternative **proposals** which were considered, the decisions made for **resolution** of the issues, the **criteria** used to guide decisions and the **arguments** developers went through to reach a decision.

Rationale (1)

- **Issues**
  - To each decision corresponds an issue that needs to be solved. Issues are usually phrased as questions: How ...?
- **Proposals / Alternatives**
  - Possible solutions that could address the issue considered. Includes alternatives that were explored but discarded.

Rationale (2)

- **Criteria**
  - Desirable qualities that the selected solution should satisfy.
  - **Requirements analysis** criteria include usability, number of input errors per day.
  - **Design** criteria include reliability, response time.
  - **Project management** criteria include trade-offs such as timely delivery vs. quality.

Rationale (3)

- **Arguments**
  - The discussions which took place in decision making as developers discover issues, try solutions, and argue their relative benefits.
- **Resolution**
  - The decision taken to resolve an issue. An alternative is selected which satisfies the criteria, supported by arguments for that decision.

Levels of rationale

- **No rationale captured**
  - Rationale is only present in memos, online communication, developers’ memory
- **Rationale reconstruction**
  - Rationale is documented in a document justifying the final design
- **Rationale capture**
  - Rationale is documented during design as it is developed
- **Rationale integration**
  - Rationale drives the design

Rationale Exercise

1. Read the excerpt from the design documents for an accident management system in B & D pg 528 (see lecture handout)
2. The excerpt presents the rationale for using a relational database for permanent storage. The argument is presented in prose. Rewrite it in terms of **issues, alternatives, arguments, criteria & resolutions**
3. Which version of the document (free prose or issue model) would be easiest to work with during, say, system maintenance? Why?
Rationale in Practice

Record and replay

- Facilitator posts an agenda
  - Discussion items are issues
- Participants respond to the agenda
  - Proposed amendments are proposals or additional issues
- Facilitator updates the agenda and facilitates the meeting
  - The scope of each discussion is a single issue tree
- Minute taker records the meeting
  - The minute taker records discussions in terms of issues, proposals, arguments, and criteria.
  - The minute taker records decisions as resolutions and action items.

Record and replay example: database discussion agenda

3. Discussion

I[1] Which policy for retrieving tracks from the database?
I[3] Which query language for specifying tracks in the database request?

Record and replay example: database discussion

I[1] Which policy for retrieving tracks from the database?
Jim: How about we just retrieve the track specified by the query? It is straightforward to implement and we can always revisit it if it is too slow.
Ann: Prefetching neighboring tracks would not be much difficult and way faster.
Sam: During route planning, we usually need the neighbor tracks anyway. Queries for route planning are the most common queries.
Jim: Ok, let’s go for the prefetch solution. We can revert to the simpler solution if it gets too complicated.

Record and replay example: database discussion minutes

3. Discussion

I[1] Which policy for retrieving tracks from the database?
  P[1.1] Single tracks!
  A- Lower throughput.
  A+ Simpler.
  P[1.2] Tracks + neighbors!
  A+ Overall better performance: during route planning, we need the neighbors anyway.
  {ref: 1/31 routing meeting}
R[1] Implement P[1.2]. However, the prefetch should be implemented in the database layer, allowing use to encapsulate this decision. If all else fails, we will fall back on P[1.1].

Open issues for Rationale

- Formalizing knowledge is costly
  - Maintaining a consistent design model is expensive.
  - Capturing and maintaining its rationale is worse.
- The benefits of rationale are not perceived by current developers
  - If the person who does the work is not the one who benefits from it, the work will have lower priority.
  - 40-90% of off-the-shelf software projects are terminated before the product ships.
- Capturing rationale is usually disruptive
- Current approaches do not scale to real problems